24 | Spark 3.0（一）：AQE的3个特性怎么才能用好?

2021-05-07 吴磊

Spark性能调优实战 进入课程 >

讲述：吴磊
时长 18:32 大小 16.98M

你好，我是吴磊。

目前，距离 Spark 3.0 版本的发布已经将近一年的时间了，这次版本升级添加了自适应查询执行（AQE）、动态分区剪裁（DPP）和扩展的 Join Hints 等新特性。利用好这些新特性，可以让我们的性能调优如虎添翼。因此，我会用三讲的时间和你聊聊它们。今天，我们先来说说 AQE。

我发现，同学们在使用 AQE 的时候总会抱怨说：“AQE 的开关打开了，相关的配置项也设了，可应用性能还是没有提升。”这往往是因为我们对于 AQE 的理解不够透彻，调优是照葫芦画瓢，所以这一讲，我们就先从 AQE 的设计初衷说起，然后说说它的工作原理，最后再去探讨怎样才能用好 AQE。

Spark 为什么需要 AQE?

在 2.0 版本之前，Spark SQL 仅仅支持启发式、静态的优化过程，就像我们在第 21、22、23 三讲介绍的一样。

启发式的优化又叫 RBO（Rule Based Optimization，基于规则的优化），它往往基于一些规则和策略实现，如谓词下推、列剪枝，这些规则和策略来源于数据库领域已有的应用经验。也就是说，**启发式的优化实际上算是一种经验主义**。

经验主义的弊端就是不分青红皂白、胡子眉毛一把抓，对待相似的问题和场景都使用同一类套路。Spark 社区正是因为意识到了 RBO 的局限性，因此在 2.2 版本中推出了 CBO（Cost Based Optimization，基于成本的优化）。

CBO 的特点是“实事求是”，基于数据表的统计信息（如表大小、数据列分布）来选择优化策略。CBO 支持的统计信息很丰富，比如数据表的行数、每列的基数（Cardinality）、空值数、最大值、最小值和直方图等等。因为有统计数据做支持，所以 CBO 选择的优化策略往往优于 RBO 选择的优化规则。

但是，**CBO 也面临三个方面的窘境：“窄、慢、静”**。窄指的是适用面太窄，CBO 仅支持注册到 Hive Metastore 的数据表，但在大量的应用场景中，数据源往往是存储在分布式文件系统的各类文件，如 Parquet、ORC、CSV 等等。

慢指的是统计信息的搜集效率比较低。对于注册到 Hive Metastore 的数据表，开发者需要调用 ANALYZE TABLE COMPUTE STATISTICS 语句收集统计信息，而各类信息的收集会消耗大量时间。

静指的是静态优化，这一点与 RBO 一样。CBO 结合各类统计信息制定执行计划，一旦执行计划交付运行，CBO 的使命就算完成了。换句话说，如果在运行时数据分布发生动态变化，CBO 先前制定的执行计划并不会跟着调整、适配。

AQE 到底是什么?

考虑到 RBO 和 CBO 的种种限制，Spark 在 3.0 版本推出了 AQE（Adaptive Query Execution，自适应查询执行）。如果用一句话来概括，**AQE 是 Spark SQL 的一种动态优化机制，在运行时，每当 Shuffle Map 阶段执行完毕，AQE 都会结合这个阶段的统计信**

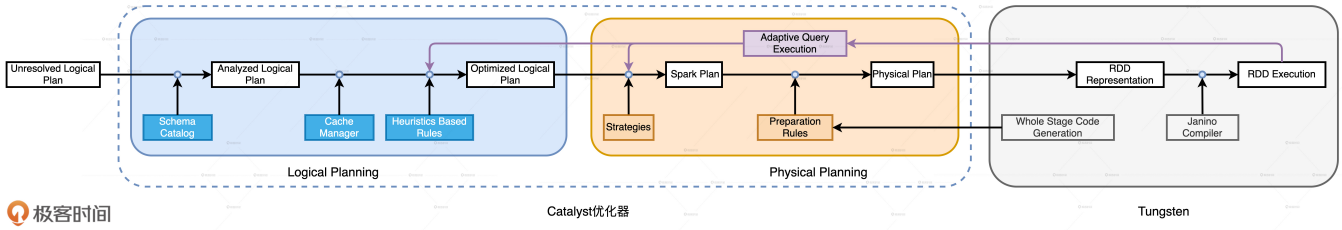
息，基于既定的规则动态地调整、修正尚未执行的逻辑计划和物理计划，来完成对原始查询语句的运行时优化。

从定义中，我们不难发现，**AQE 优化机制触发的时机是 Shuffle Map 阶段执行完毕。也就是说，AQE 优化的频次与执行计划中 Shuffle 的次数一致。**反过来说，如果你的查询语句不会引入 Shuffle 操作，那么 Spark SQL 是不会触发 AQE 的。对于这样的查询，无论你怎么调整 AQE 相关的配置项，AQE 也都爱莫能助。

对于 AQE 的定义，我相信你还有很多问题，比如，AQE 依赖的统计信息具体是什么？既定的规则和策略具体指什么？接下来，我们一一来解答。

首先，AQE 赖以优化的统计信息与 CBO 不同，这些统计信息并不是关于某张表或是哪个列，而是 Shuffle Map 阶段输出的中间文件。学习过 Shuffle 的工作原理之后，我们知道，每个 Map Task 都会输出以 data 为后缀的数据文件，还有以 index 为结尾的索引文件，这些文件统称为中间文件。每个 data 文件的大小、空文件数量与占比、每个 Reduce Task 对应的分区大小，所有这些基于中间文件的统计值构成了 AQE 进行优化的信息来源。

其次，结合 Spark SQL 端到端优化流程图我们可以看到，AQE 从运行时获取统计信息，在条件允许的情况下，优化决策会分别作用到逻辑计划和物理计划。



AQE在Spark SQL中的位置与作用

AQE 既定的规则和策略主要有 4 个，分为 1 个逻辑优化规则和 3 个物理优化策略。我把这些规则与策略，和相应的 AQE 特性，以及每个特性仰仗的统计信息，都汇总到了如下的表格中，你可以看一看。

优化类型	规则与策略	AQE特性	统计信息
逻辑计划	DemoteBroadcastHashJoin	Join策略调整	Map阶段中间文件总大小 中间文件空文件占比
物理计划	OptimizeLocalShuffleReader		
	CoalesceShufflePartitions	自动分区合并	每个Reduce Task分区大小
	OptimizeSkewedJoin	自动倾斜处理	每个Reduce Task分区大小

极客时间

如何用好 AQE?

那么，AQE 是如何根据 Map 阶段的统计信息以及这 4 个规则与策略，来动态地调整和修正尚未执行的逻辑计划和物理计划的呢？这就要提到 AQE 的三大特性，也就是 Join 策略调整、自动分区合并，以及自动倾斜处理，我们需要借助它们去分析 AQE 动态优化的过程。它们的基本概念我们在第 9 讲说过，这里我再带你简单回顾一下。

> Join 策略调整：如果某张表在过滤之后，尺寸小于广播变量阈值，这张表参与的数据关联就会从 Shuffle Sort Merge Join 降级（Demote）为执行效率更高的 Broadcast Hash Join。
>
> 自动分区合并：在 Shuffle 过后，Reduce Task 数据分布参差不齐，AQE 将自动合并过小的数据分区。
>
> 自动倾斜处理：结合配置项，AQE 自动拆分 Reduce 阶段过大的数据分区，降低单个 Reduce Task 的工作负载。

接下来，我们就一起来分析这 3 个特性的动态优化过程。

Join 策略调整

我们先来说说 Join 策略调整，这个特性涉及了一个逻辑规则和一个物理策略，它们分别是 DemoteBroadcastHashJoin 和 OptimizeLocalShuffleReader。

DemoteBroadcastHashJoin 规则的作用，是把 Shuffle Joins 降级为 Broadcast Joins。需要注意的是，这个规则仅适用于 Shuffle Sort Merge Join 这种关联机制，其

他机制如 Shuffle Hash Join、Shuffle Nested Loop Join 都不支持。对于参与 Join 的两张表来说，在它们分别完成 Shuffle Map 阶段的计算之后，DemoteBroadcastHashJoin 会判断中间文件是否满足如下条件：

> 中间文件尺寸总和小于广播阈值 spark.sql.autoBroadcastJoinThreshold
>
> 空文件占比小于配置项 spark.sql.adaptive.nonEmptyPartitionRatioForBroadcastJoin

只要有任意一张表的统计信息满足这两个条件，Shuffle Sort Merge Join 就会降级为 Broadcast Hash Join。说到这儿，你可能会问：“既然 DemoteBroadcastHashJoin 逻辑规则可以把 Sort Merge Join 转换为 Broadcast Join，那同样用来调整 Join 策略的 OptimizeLocalShuffleReader 规则又是干什么用的呢？看上去有些多余啊！”

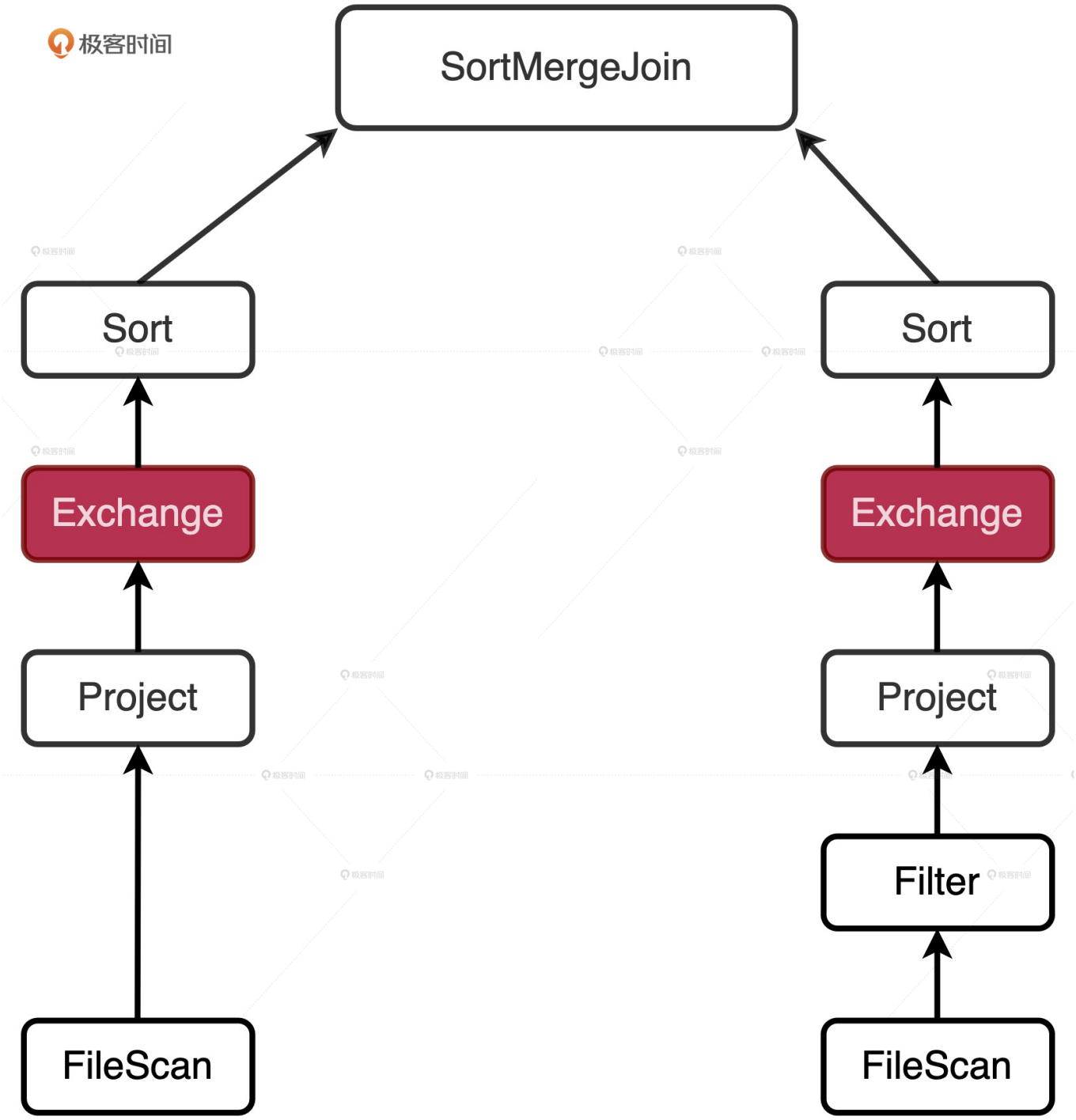
不知道你注意到没有，我一直强调，**AQE 依赖的统计信息来自于 Shuffle Map 阶段生成的中间文件。**这意味什么呢？这就意味着 AQE 在开始优化之前，Shuffle 操作已经执行过半了！

我们来举个例子，现在有两张表：事实表 Order 和维度表 User，它们的查询语句和初始的执行计划如下。

复制代码

```
//订单表与用户表关联
select sum(order.price * order.volume), user.id
from order inner join user
on order.userId = user.id
where user.type = ‘Head Users’
group by user.id
```

由于两张表大都到超过了广播阈值，因此 Spark SQL 在最初的执行计划中选择了 Sort Merge Join。AQE 需要同时结合两个分支中的 Shuffle（Exchange）输出，才能判断是否可以降级为 Broadcast Join，以及用哪张表降级。这就意味着，不论大表还是小表都要完成 Shuffle Map 阶段的计算，并且把中间文件落盘，AQE 才能做出决策。



Sort Merge Join执行计划（左侧是事实表的执行分支，右侧是维度表的执行分支）

你可能会说：“根本不需要大表做 Shuffle 呀，AQE 只需要去判断小表 Shuffle 的中间文件就好啦”。可问题是，AQE 可分不清哪张是大表、哪张是小表。在 Shuffle Map 阶段结束之前，数据表的尺寸大小对于 AQE 来说是“透明的”。因此，AQE 必须等待两张表都完成 Shuffle Map 的计算，然后统计中间文件，才能判断降级条件是否成立，以及用哪张表做广播变量。

在常规的 Shuffle 计算流程中，Reduce 阶段的计算需要跨节点访问中间文件拉取数据分片。如果遵循常规步骤，即便 AQE 在运行时把 Shuffle Sort Merge Join 降级为

Broadcast Join，大表的中间文件还是需要通过网络进行分发。这个时候，AQE 的动态 Join 策略调整也就失去了实用价值。原因很简单，负载最重的大表 Shuffle 计算已经完成，再去决定切换到 Broadcast Join 已经没有任何意义。

在这样的背景下，OptimizeLocalShuffleReader 物理策略就非常重要了。既然大表已经完成 Shuffle Map 阶段的计算，这些计算可不能白白浪费掉。**采取 OptimizeLocalShuffleReader 策略可以省去 Shuffle 常规步骤中的网络分发，Reduce Task 可以就地读取本地节点（Local）的中间文件，完成与广播小表的关联操作。**

不过，需要我们特别注意的是，**OptimizeLocalShuffleReader 物理策略的生效与否由一个配置项决定**。这个配置项是 spark.sql.adaptive.localShuffleReader.enabled，尽管它的默认值是 True，但是你千万不要把它的值改为 False。否则，就像我们刚才说的，AQE 的 Join 策略调整就变成了形同虚设。

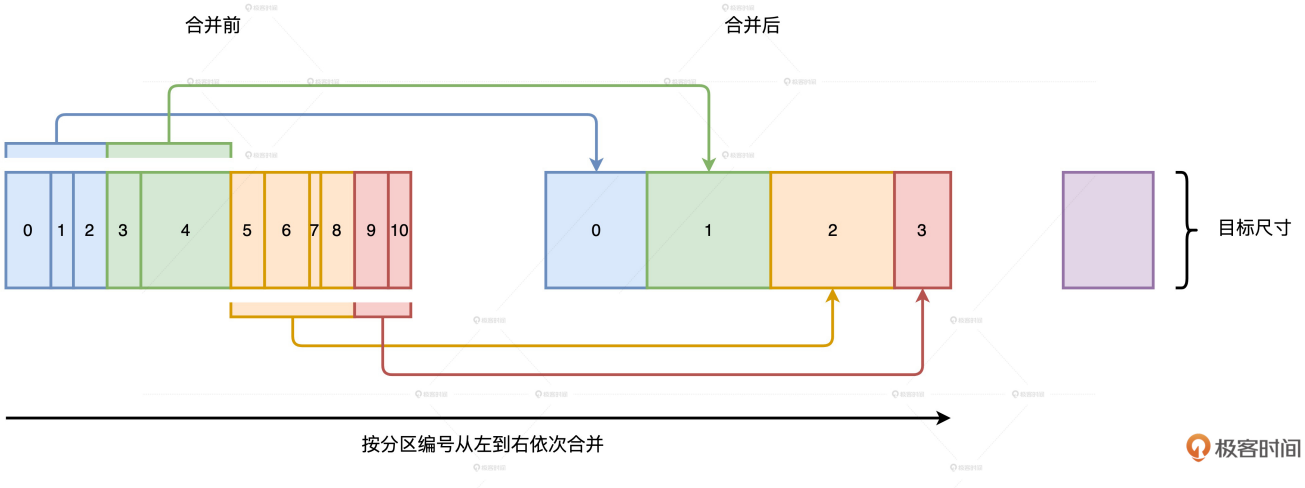
说到这里，你可能会说：“这么看，AQE 的 Join 策略调整有些鸡肋啊！毕竟 Shuffle 计算都已经过半，Shuffle Map 阶段的内存消耗和磁盘 I/O 是半点没省！”确实，Shuffle Map 阶段的计算开销是半点没省。但是，OptimizeLocalShuffleReader 策略避免了 Reduce 阶段数据在网络中的全量分发，仅凭这一点，大多数的应用都能获益匪浅。因此，对于 AQE 的 Join 策略调整，**我们可以用一个成语来形容：“亡羊补牢、犹未为晚”**。

自动分区合并

接下来，我们再来说说自动分区合并。分区合并的原理比较简单，**在 Reduce 阶段，当 Reduce Task 从全网把数据分片拉回，AQE 按照分区编号的顺序，依次把小于目标尺寸的分区合并在一起**。目标分区尺寸由以下两个参数共同决定。这部分我们在第 10 讲详细讲过，如果不记得，你可以翻回去看一看。

spark.sql.adaptive.advisoryPartitionSizeInBytes，由开发者指定分区合并后的推荐尺寸。

spark.sql.adaptive.coalescePartitions.minPartitionNum，分区合并后，分区数不能低于该值。



除此之外，我们还要注意，在 Shuffle Map 阶段完成之后，AQE 优化机制被触发，CoalesceShufflePartitions 策略“无条件”地被添加到新的物理计划中。读取配置项、计算目标分区大小、依序合并相邻分区这些计算逻辑，在 Tungsten WSCG 的作用下融合进“手写代码”于 Reduce 阶段执行。

自动倾斜处理

与自动分区合并相反，自动倾斜处理的操作是“拆”。在 Reduce 阶段，当 Reduce Task 所需处理的分区尺寸大于一定阈值时，利用 OptimizeSkewedJoin 策略，AQE 会把大分区拆成多个小分区。倾斜分区和拆分粒度由以下这些配置项决定。关于它们的含义与作用，我们在第 10 讲说过，你可以再翻回去看一看。

spark.sql.adaptive.skewJoin.skewedPartitionFactor，判定倾斜的膨胀系数

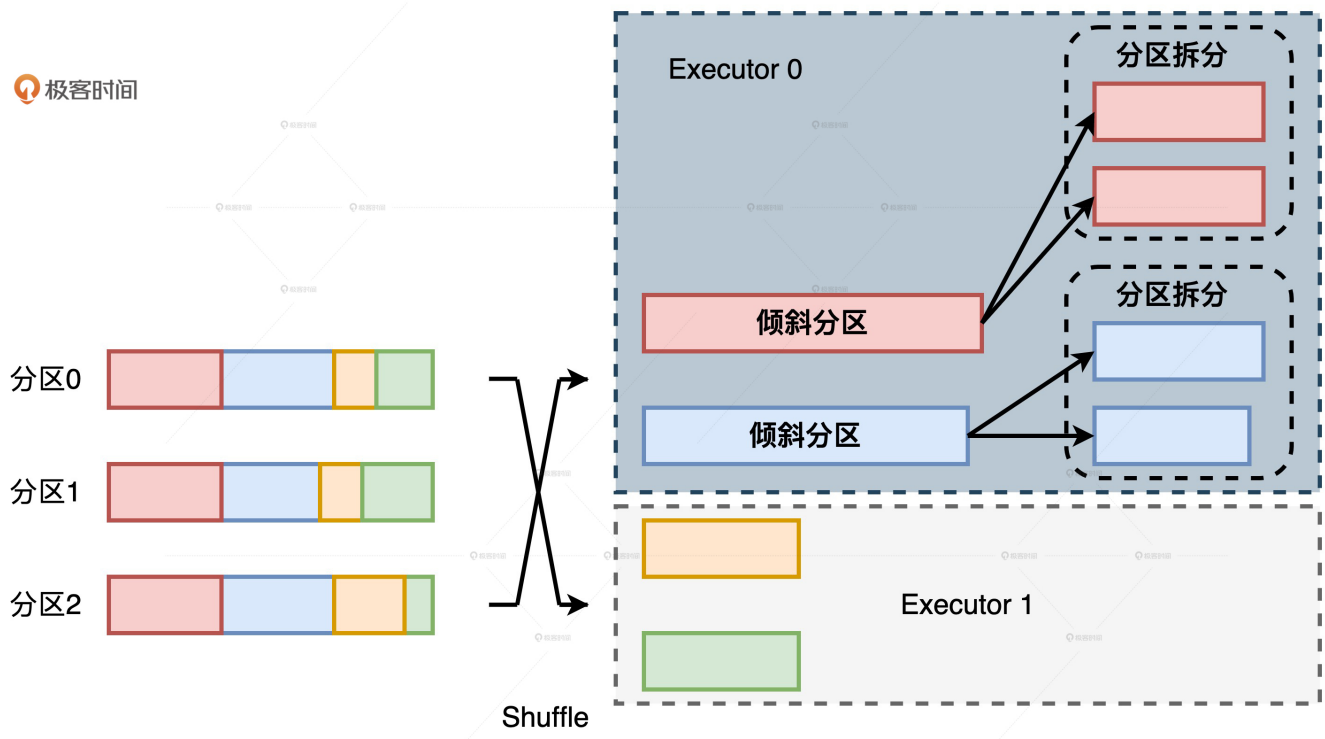
spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes，判定倾斜的最低阈值

spark.sql.adaptive.advisoryPartitionSizeInBytes，以字节为单位，定义拆分粒度

自动倾斜处理的拆分操作也是在 Reduce 阶段执行的。在同一个 Executor 内部，本该由一个 Task 去处理的大分区，被 AQE 拆成多个小分区并交由多个 Task 去计算。这样一来，Task 之间的计算负载就可以得到平衡。但是，这并不能解决不同 Executors 之间的负载均衡问题。

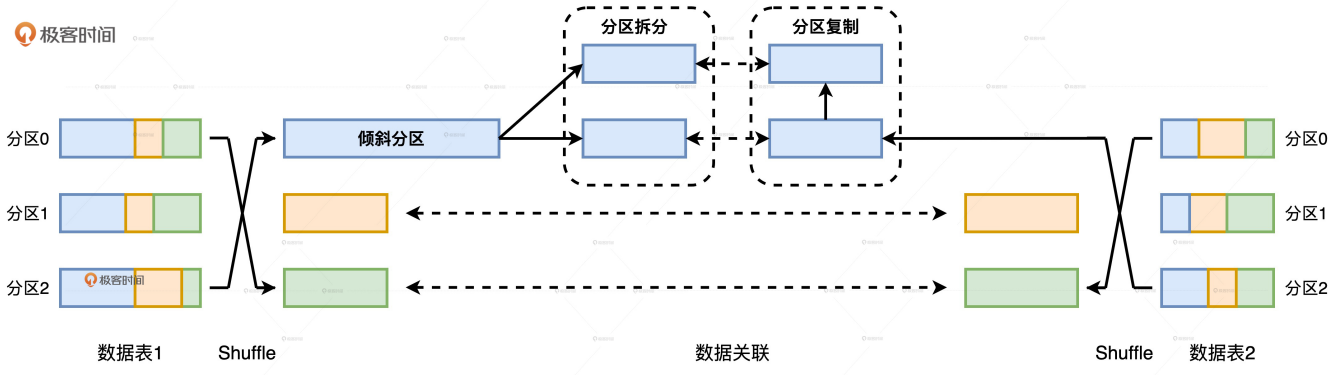
我们来举个例子，假设有个 Shuffle 操作，它的 Map 阶段有 3 个分区，Reduce 阶段有 4 个分区。4 个分区中的两个都是倾斜的大分区，而且这两个倾斜的大分区刚好都分发到了 Executor 0。通过下图，我们能够直观地看到，尽管两个大分区被拆分，但横向来看，整

个作业的主要负载还是落在了 Executor 0 的身上。Executor 0 的计算能力依然是整个作业的瓶颈，这一点并没有因为分区拆分而得到实质性的缓解。



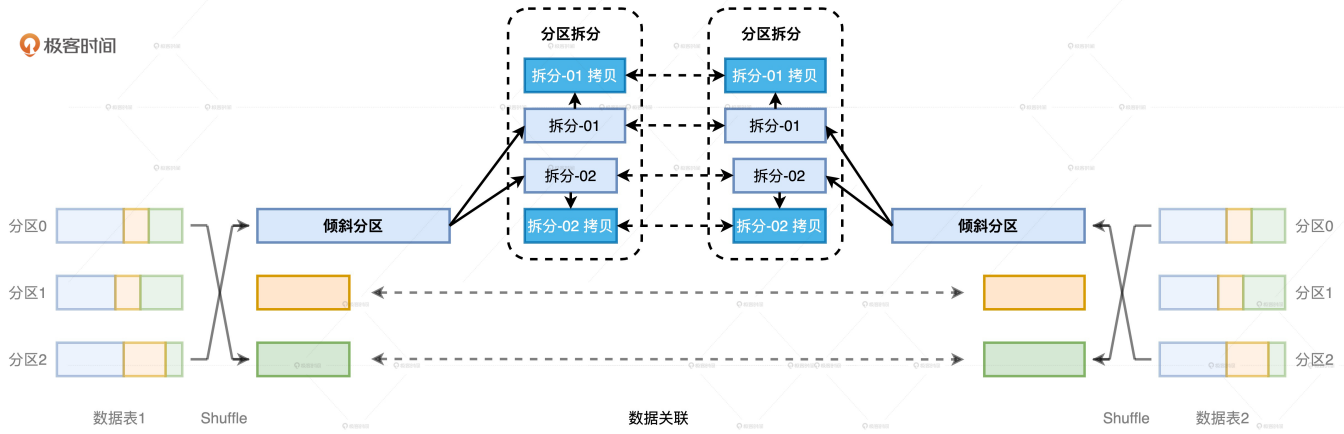
Executors之间的负载平衡还有待优化

另外，在数据关联的场景中，对于参与 Join 的两张表，我们暂且把它们记做数据表 1 和数据表 2，如果表 1 存在数据倾斜，表 2 不倾斜，那在关联的过程中，AQE 除了对表 1 做拆分之外，还需要对表 2 对应的数据分区做复制，来保证关联关系不被破坏。



数据关联场景中的自动倾斜处理

在这样的运行机制下，如果两张表都存在数据倾斜怎么办？这个时候，事情就开始变得逐渐复杂起来了。对于上图中的表 1 和表 2，我们假设表 1 还是拆出来两个分区，表 2 因为倾斜也拆出来两个分区。这个时候，为了不破坏逻辑上的关联关系，表 1、表 2 拆分出来的分区还要各自复制出一份，如下图所示。



两边倾斜

如果现在问题变得更复杂了，左表拆出 M 个分区，右表拆出 N 各分区，那么每张表最终都需要保持 M x N 份分区数据，才能保证关联逻辑的一致性。当 M 和 N 逐渐变大时，AQE 处理数据倾斜所需的计算开销将会面临失控的风险。

总的来说，当应用场景中的数据倾斜比较简单，比如虽然有倾斜但数据分布相对均匀，或是关联计算中只有一边倾斜，我们完全可以依赖 AQE 的自动倾斜处理机制。但是，当我们的场景中数据倾斜变得复杂，比如数据中不同 Key 的分布悬殊，或是参与关联的两表都存在大量的倾斜，我们就需要衡量 AQE 的自动化机制与手工处理倾斜之间的利害得失。关于手工处理倾斜，我们留到第 28 讲再去展开。

小结

AQE 是 Spark SQL 的一种动态优化机制，它的诞生解决了 RBO、CBO，这些启发式、静态优化机制的局限性。想要用好 AQE，我们就要掌握它的特点，以及它支持的三种优化特性的工作原理和使用方法。

如果用一句话来概括 AQE 的定义，就是每当 Shuffle Map 阶段执行完毕，它都会结合这个阶段的统计信息，根据既定的规则和策略动态地调整、修正尚未执行的逻辑计划和物理计划，从而完成对原始查询语句的运行时优化。也因此，只有当你的查询语句会引入 Shuffle 操作的时候，Spark SQL 才会触发 AQE。

AQE 支持的三种优化特性分别是 Join 策略调整、自动分区合并和自动倾斜处理。

关于 Join 策略调整，我们首先要知道 DemoteBroadcastHashJoin 规则仅仅适用于 Shuffle Sort Merge Join 这种关联机制，对于其他 Shuffle Joins 类型，AQE 暂不支持把

它们转化为 Broadcast Joins。其次，为了确保 AQE 的 Join 策略调整正常运行，我们要确保 spark.sql.adaptive.localShuffleReader.enabled 配置项始终为开启状态。

关于自动分区合并，我们要知道，在 Shuffle Map 阶段完成之后，结合分区推荐尺寸与分区数量限制，AQE 会自动帮我们完成分区合并的计算过程。

关于 AQE 的自动倾斜处理我们要知道，它只能以 Task 为粒度缓解数据倾斜，并不能解决不同 Executors 之间的负载均衡问题。针对场景较为简单的倾斜问题，比如关联计算中只涉及单边倾斜，我们完全可以依赖 AQE 的自动倾斜处理机制。但是，当数据倾斜问题变得复杂的时候，我们需要衡量 AQE 的自动化机制与手工处理倾斜之间的利害得失。

每日一练

1. 我们知道，AQE 依赖的统计信息来源于 Shuffle Map 阶段输出的中间文件。你觉得，在运行时，AQE 还有其他渠道可以获得同样的统计信息吗？
2. AQE 的自动倾斜处理机制只能以 Task 为粒度来平衡工作负载，如果让你重新实现这个机制，你有什么更好的办法能让 AQE 以 Executors 为粒度做到负载均衡吗？

期待在留言区看到你的思考和答案，我们下一讲见！

提建议

学习推荐

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 23 | 钨丝计划：Tungsten给开发者带来了哪些福报?

下一篇 25 | Spark 3.0（二）：DPP特性该怎么用?

精选留言 (4)

写留言

Fendora范东_
2021-05-09

磊哥，请教下
Join策略调整里面，为啥只有SMJ能降级，而其他Join比如SHJ不能降级

展开

作者回复: 好问题~

其实这个问题我特别想让你独立思考，你先花5分钟想想，结合咱们之前讲过的内容，然后再看答复~

————————————给你留个分割线，哈哈————————————

首先，Spark有个参数：spark.sql.join.prefersortmergeJoin，它默认是true，也就是说，对于Shuffle Joins，Spark SQL会prefer也就是偏向优先选择Shuffle Sort Merge Join，换句话说，如

果你不动这个参数的话，那么Spark SQL根本不会去考虑用Shuffle Hash Join。这么说吧，在Spark SQL眼里，Shuffle Hash Join的地位非常的低，虽然这一点我不认可，但是Spark就是这么干的。

所以，如果你想启用Shuffle Hash Join，那就必须把spark.sql.join.prefersortmergeJoin置为false。这还不算完，即便如此，Spark SQL其实还是会倾向使用Shuffle Sort Merge Join。当且仅当如下条件成立的时候，Spark才有可能考虑Shuffle Hash Join：
1）外表比内表至少大3倍
2）内表所有数据分片，都要小于广播变量

你说说，这个条件苛刻不？因此，想要启用Shuffle Hash Join，开发者就只剩下一条路可走：Join hints。也就是使用.hint("SHUFFLE_HASH")关键字，来强制Spark SQL选择Shuffle Hash Join。但是这个时候，你都使用强制机制了，Spark SQL会走单独的code path，不用再去利用AQE的Join策略调整、去看看能不能转Broadcast Joins。

因此，总结下来，在AQE的优化机制下，只有SMJ会降级为Broadcast Hash Join，压根跟Shuffle Hash Join没啥关系~

2 2

zxk
2021-05-07

问题一：个人认为 AQE 可以在加载读取文件的时候获取一些运行时信息，或者做 cache 的时候。这里也有个疑问，就是 AQE 会不会根据这些信息也进行一些优化?
问题二想到两种方法，不知道是否可行：
1. Spark 有一个动态分配调整 executor 的功能，在 Shuffle Map 阶段由 Driver 端汇聚信息决定好倾斜数据的切割方式，之后部分数据发送到原有的 executor 上，切割出来的...
展开

作者回复: 先来说说你提的问题，好问题~

这里面涉及到一个多表Join的执行过程问题，先说结论，无论你的查询涉及多少张表，所有的数据库引擎（包括Spark SQL在内），都是以两表Join为粒度，逐渐地完成Join的。以你的例子来说，a、b、c三张表，a、c先关联，那么这个时候，a、c会生成关联后的中间结果tmp，后续和b表关联的，是这个“tmp表”，已经不再是你说的c表了。因此，回答你的问题，a、c关联利用AQE转化为BHJ，那么后续Spark SQL需要继续关联tmp和b，这个时候，AQE会继续判定tmp结果集大小，如果小于广播阈值，那么它就会把tmp与b的关联，继续转化为BHJ。相反，如果tmp很大，那么Spark SQL就用默认的SMJ。

再回过头来说问题一，现有的AQE机制，是不会利用你说的那些信息的，不过这正是问题所在，就是AQE可不可以、应不应该利用其它的运行时信息，我个人认为，是可以的，只不过这里还是

要平衡一个运行时信息统计效率的问题，这个是关键所在~

关于问题二，你说的特性，应该是Spark的Dynamic allocation，也就是以Executors为粒度，根据任务的计算负载，动态缩放Spark的分布式集群。这个确实是个不错的思路，不过呢，如果仅仅是依赖于集群的缩放，那么其实还是解决不了以task为粒度做负载均衡的问题，因为不论你有多少Executors，倾斜的数据，还是会分发到某一个Executor上面去。这块的答案其实是“两阶段Shuffle”，第一阶段通过加盐，把数据打散，这样数据就会均衡地分发到集群内所有的Executors，解决了Executors之间的负载均衡问题；第二阶段是去盐化，也就是把第一阶段添加的随机“盐粒”去掉，保证关联关系的一致性。关键两阶段Shuffle的细节，可以参考第29讲哈~

1

Stony.修行僧
2021-05-07

老师请教一个问题，一个10g的csv 文件，里面有4个字段，其中三个字段需要做匿名化。在匿名化三个字段里过程中，可以partition第四个字段来提高性能吗？求老师的意见和建议

展开

作者回复: 我先说说对于问题的理解哈，不确定我的理解是不是对的。

你的诉求：一个csv格式的文件，4个字段，目标是对其中3个字段做匿名化，想问按照第四个字段做分布式计算来提高执行效率行不行。

如果是这样的话，那我觉得完全没问题，如果第四个字段分布比较均匀，不存在明显的倾斜问题，我觉得完全可以啊，而且就是应该这么做来提升执行效率。

如果有倾斜的话，你还可以手工加盐处理，手工加盐的方法我们29讲会细说，到时候可以重点关注下~

1 1

aof
2021-05-14

1. 如果不用Map阶段的输出文件的话，那应该就是实时统计了吧，比如"ANALYZE TABLE COMPUTE STATISTICS"
2. 为了防止倾斜分区都出现在同一个Executor的情况，可以考虑对倾斜数据的key进行加前缀，然后再将这些数据进行一下重分区repartition()，分区数指定为executor的个数。但是，由于使用了repartition()，也就引入了shuffle开销，这个也是一个要平衡的问题

展开

作者回复: 第二题满分💯，没错，要实现Executors级别的均衡，那确实需要加盐来处理，也就是把Join Keys打散，这样哈希过后再分发数据，数据在Executors中的分布就会比较均衡。

第一题实时统计的思路也没问题~ 这个思路其实就是传统DBMS的真正的CBO，实时计算每一步的costs，然后按照costs动态做优化。